Bias-Variance and Cross Validation

Nipun Batra and teaching staff

IIT Gandhinagar

August 14, 2025

Table of Contents

Introduction to Bias-Variance

What is the Bias-Variance Tradeoff?

Important: The Central Challenge in Machine Learning

Every ML model faces a fundamental tension:

- Make simple assumptions $\rightarrow$ Miss important patterns (High Bias)
- Make complex assumptions $\rightarrow$ Overfit to noise (High Variance)

A Real-World Analogy: Weather Prediction

Example: Simple Model: "Tomorrow = Today"

High Bias: Ignores weather patterns
Low Variance: Always makes same type of prediction

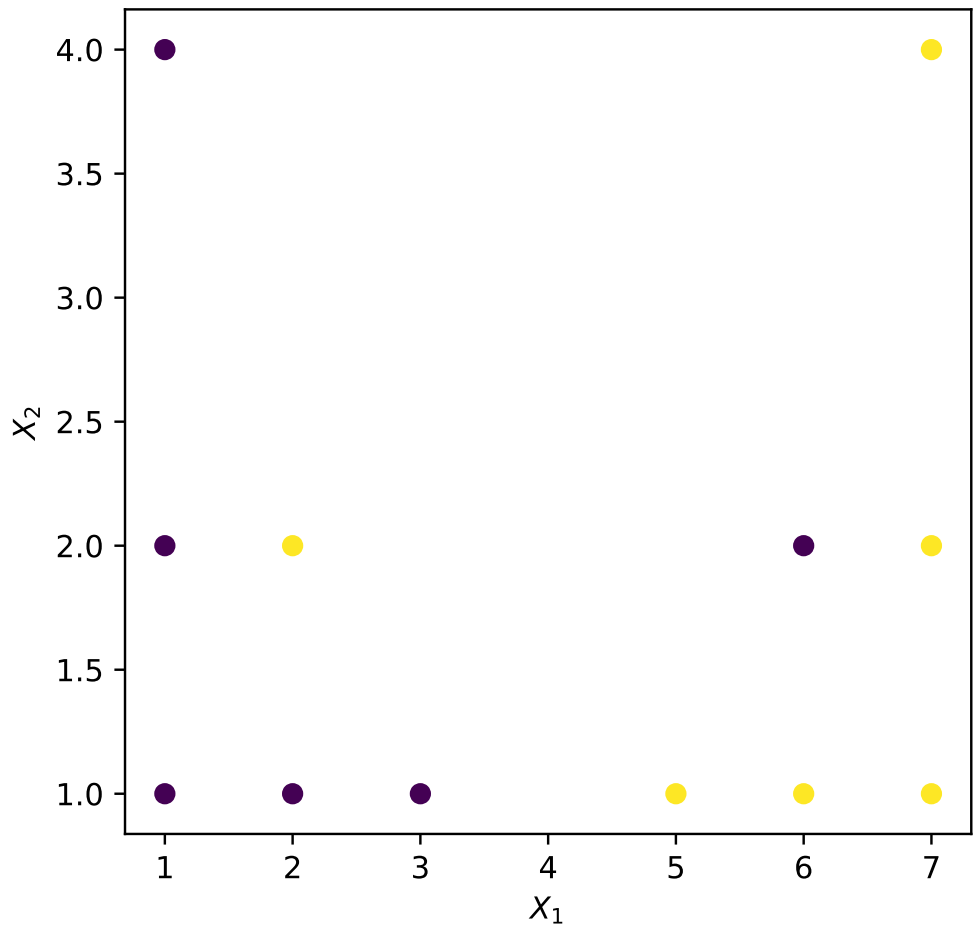
Example: Complex Model: 1000+ Variables

Low Bias: Can capture complex patterns
High Variance: Small errors $\rightarrow$ wildly different forecasts

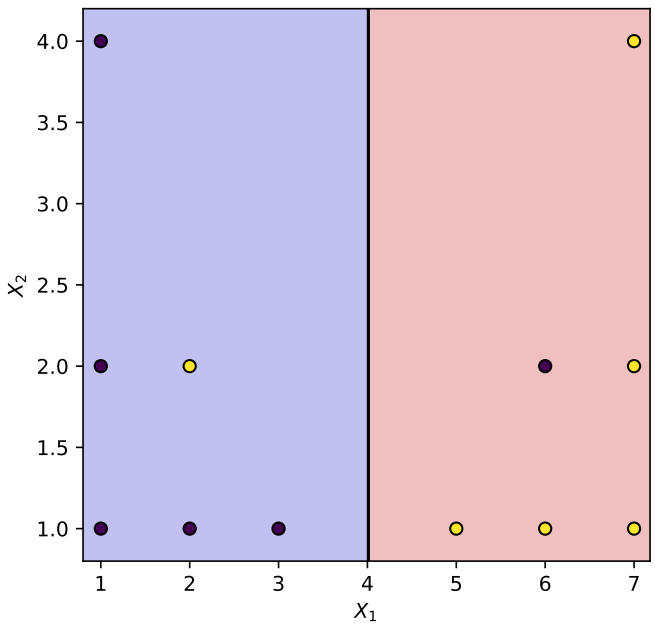
Goal: Find the sweet spot between these extremes

A Question!

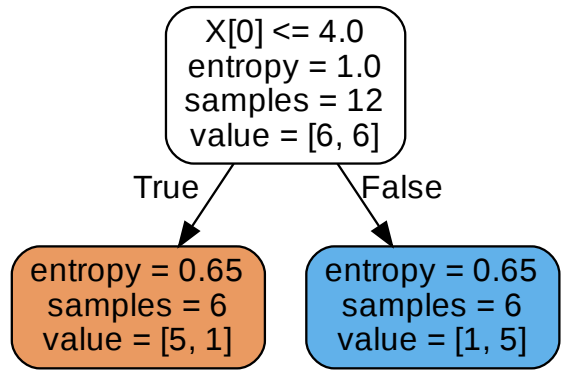
What would be the decision boundary of a decision tree classifier?



Decision Boundary for a tree with depth 1

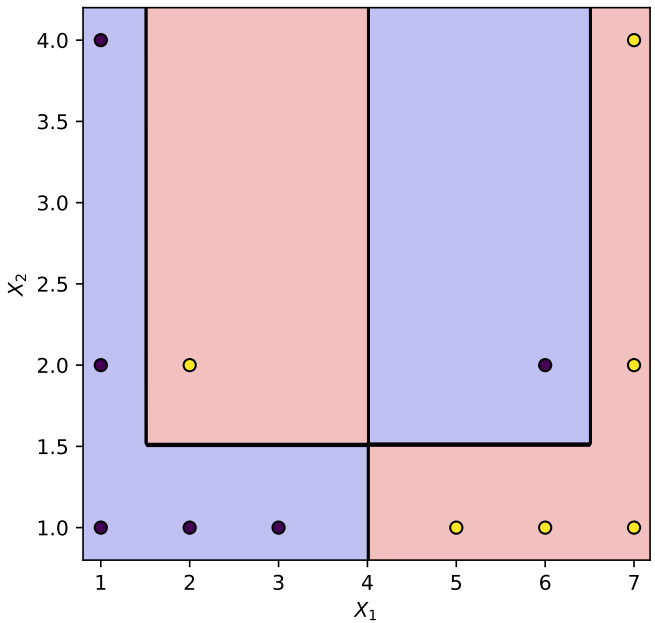


Decision Boundary

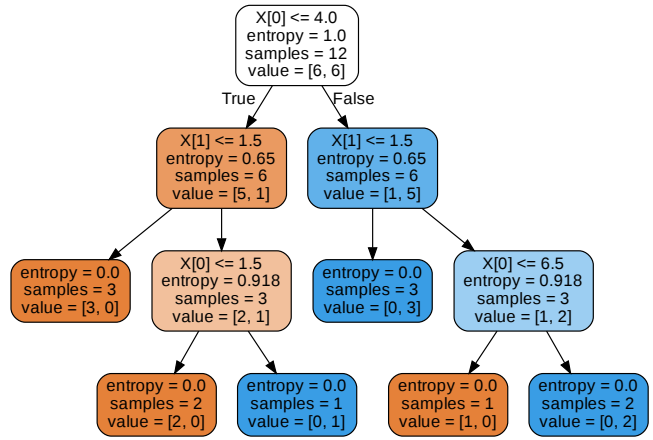


Decision Tree

Decision Boundary for a tree with no depth limit



Decision Boundary



Decision Tree

Are deeper trees always better?

As we saw, deeper trees learn more complex decision boundaries.

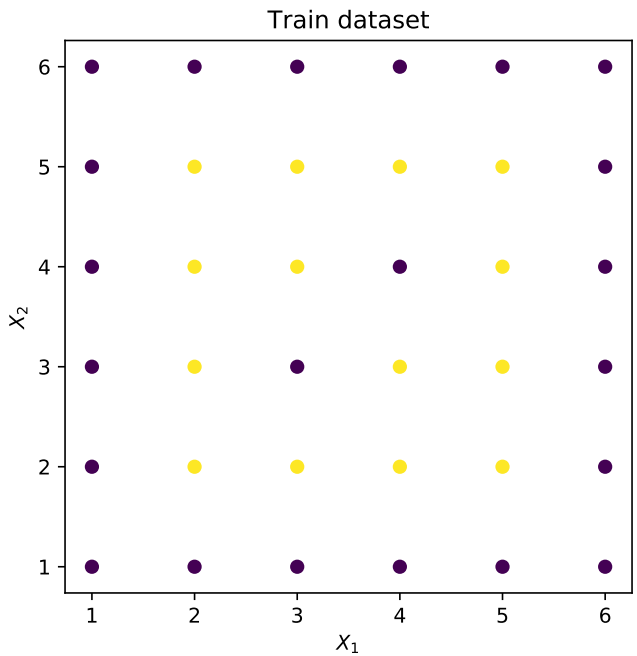
Are deeper trees always better?

As we saw, deeper trees learn more complex decision boundaries.

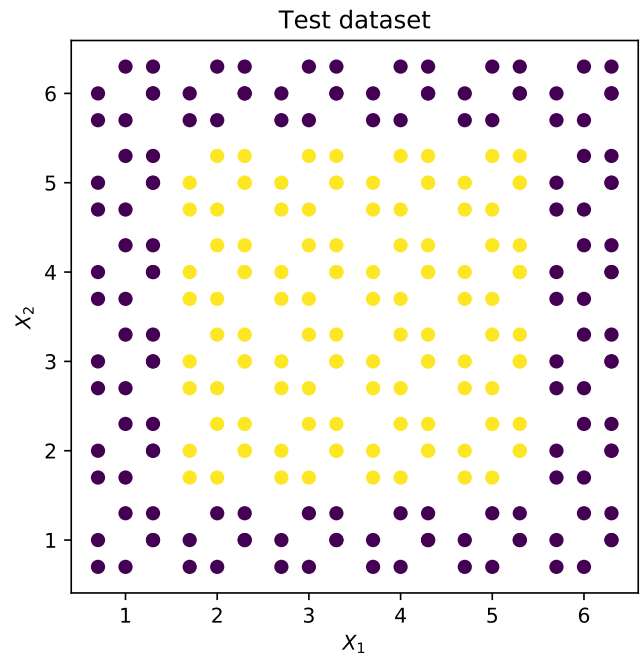
But, sometimes this can lead to poor generalization

An example

Consider the dataset below



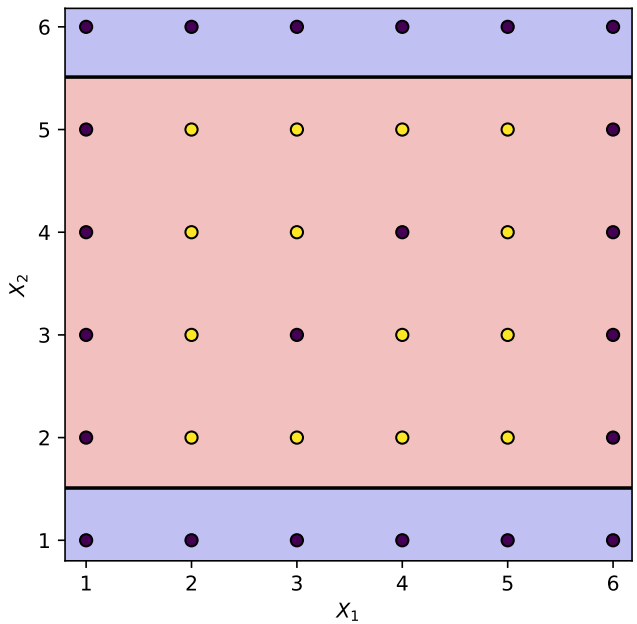
Train Set



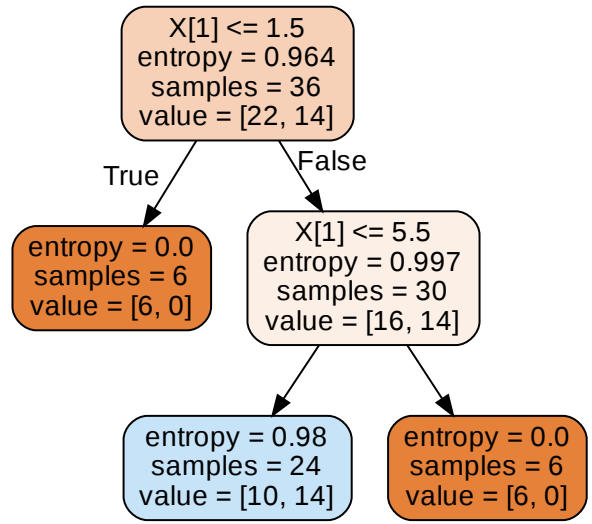
Test Set

Underfitting

Underfitting is also known as high bias, since it has a very biased incorrect assumption.



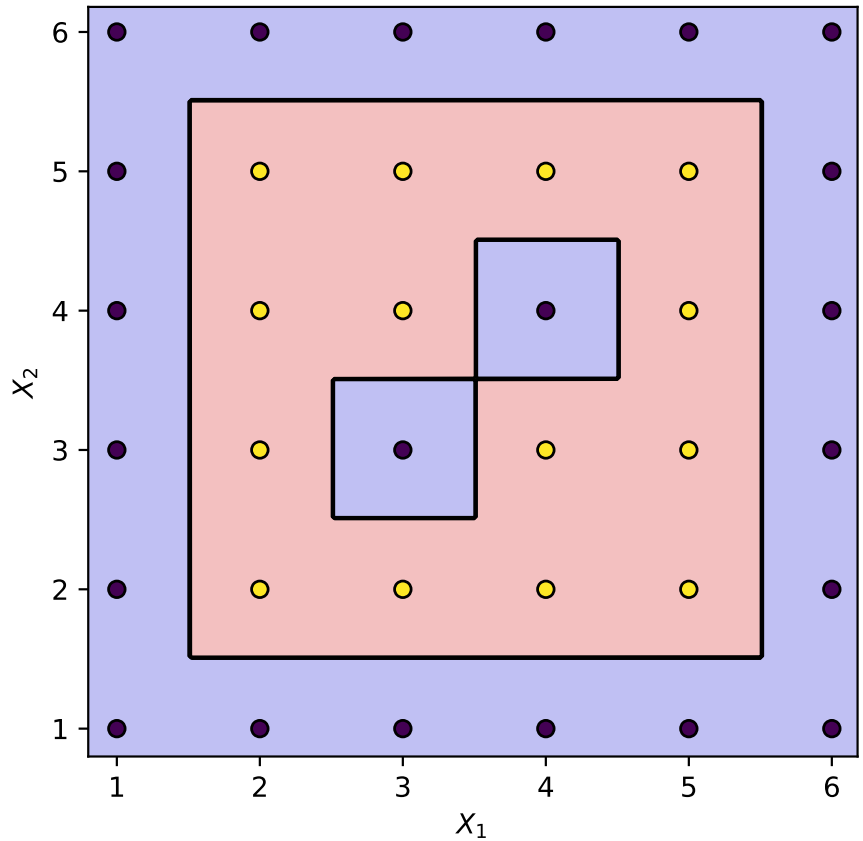
Decision Boundary



Decision Tree

Overfitting

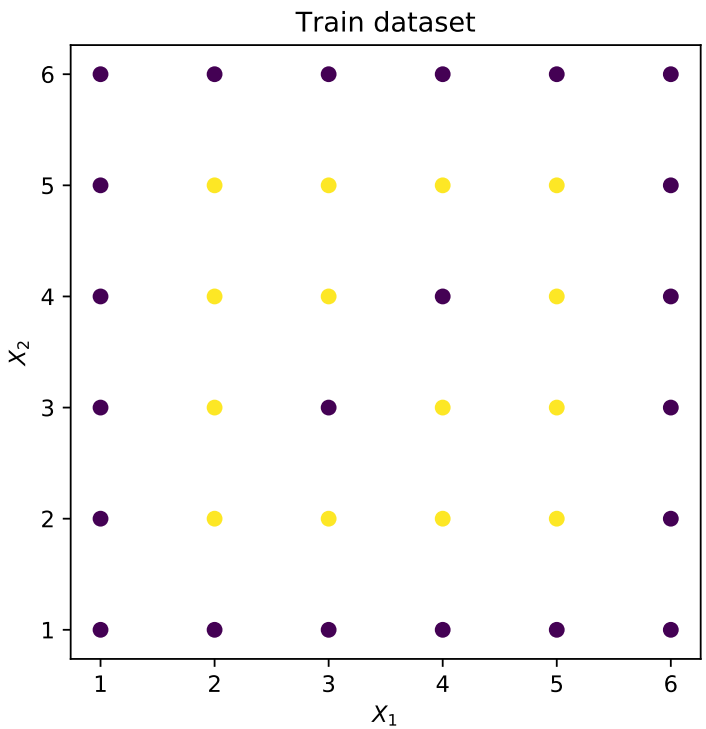
Overfitting is also known as high variance, since very small changes in data can lead to very different models.
Decision tree learned has depth of 10.



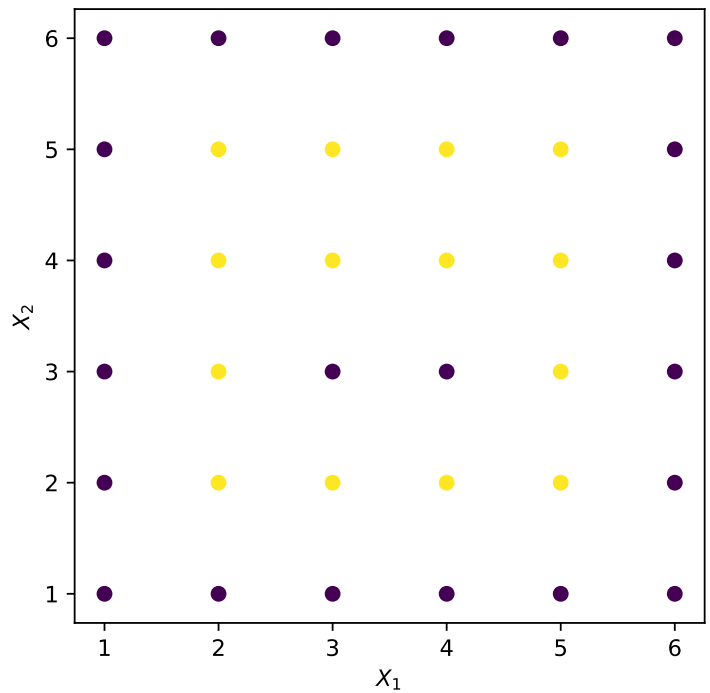
Intuition for Variance

A small change in data can lead to very different models.

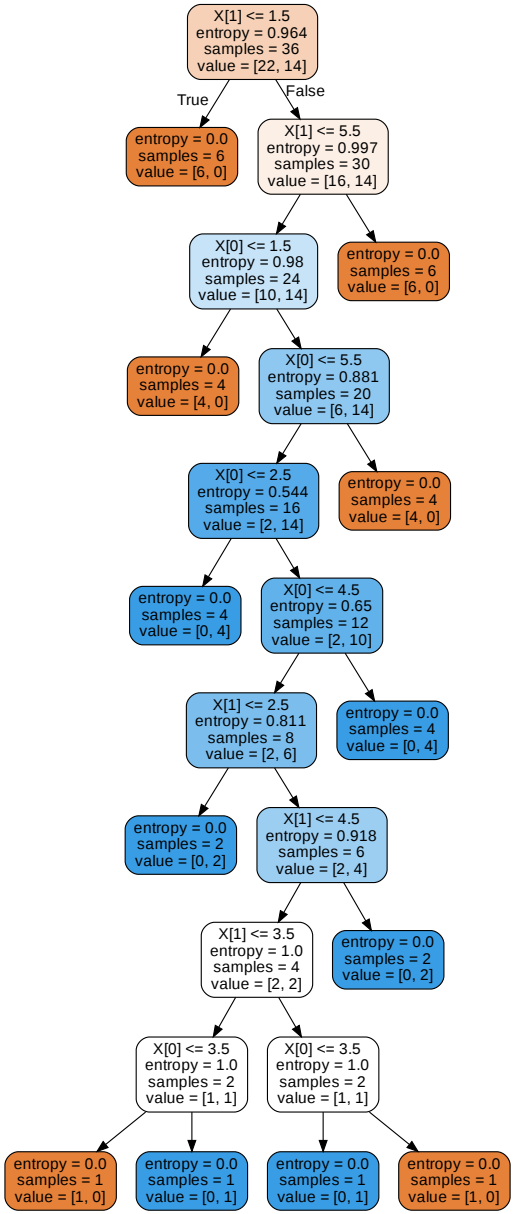
Dataset 1

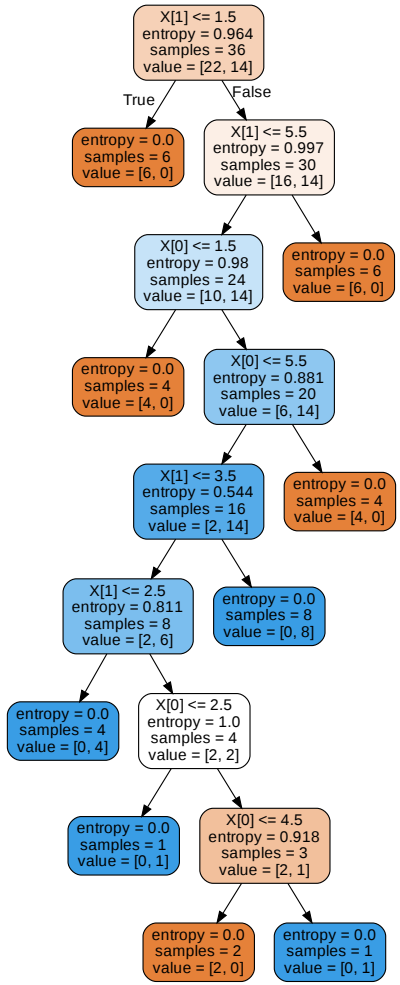


Dataset 2

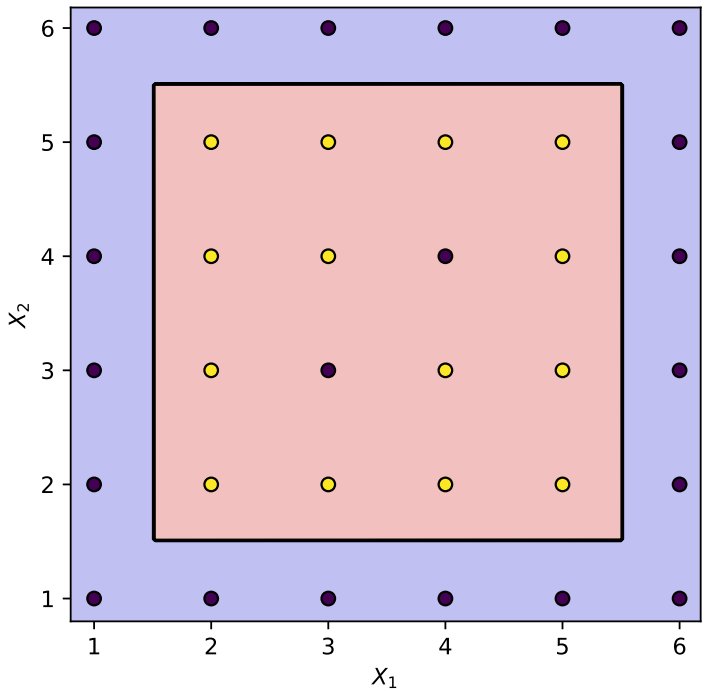


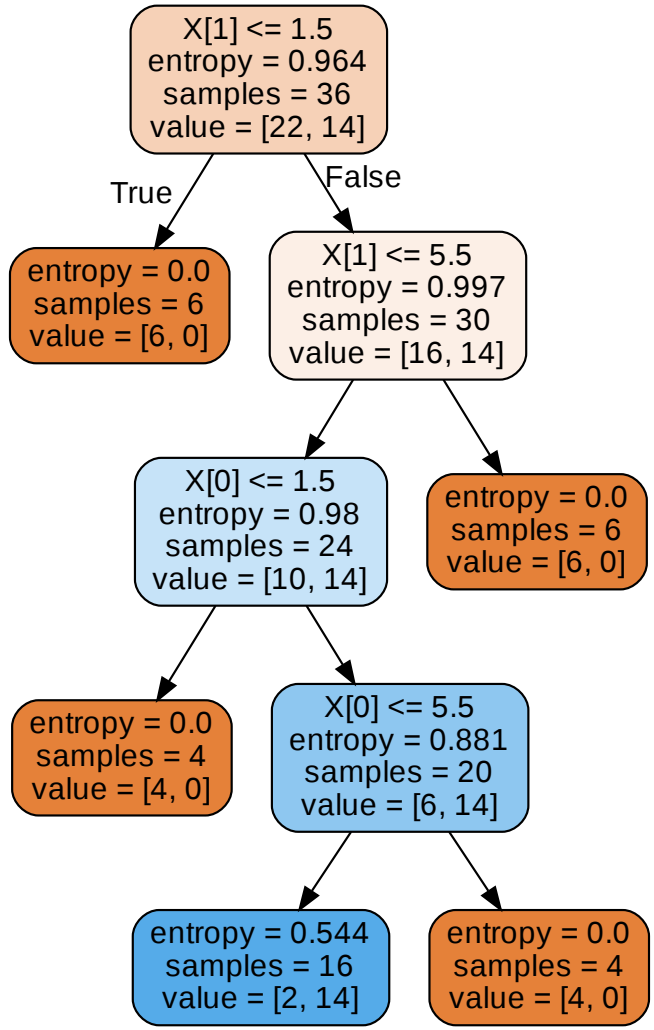
Intuition for Variance



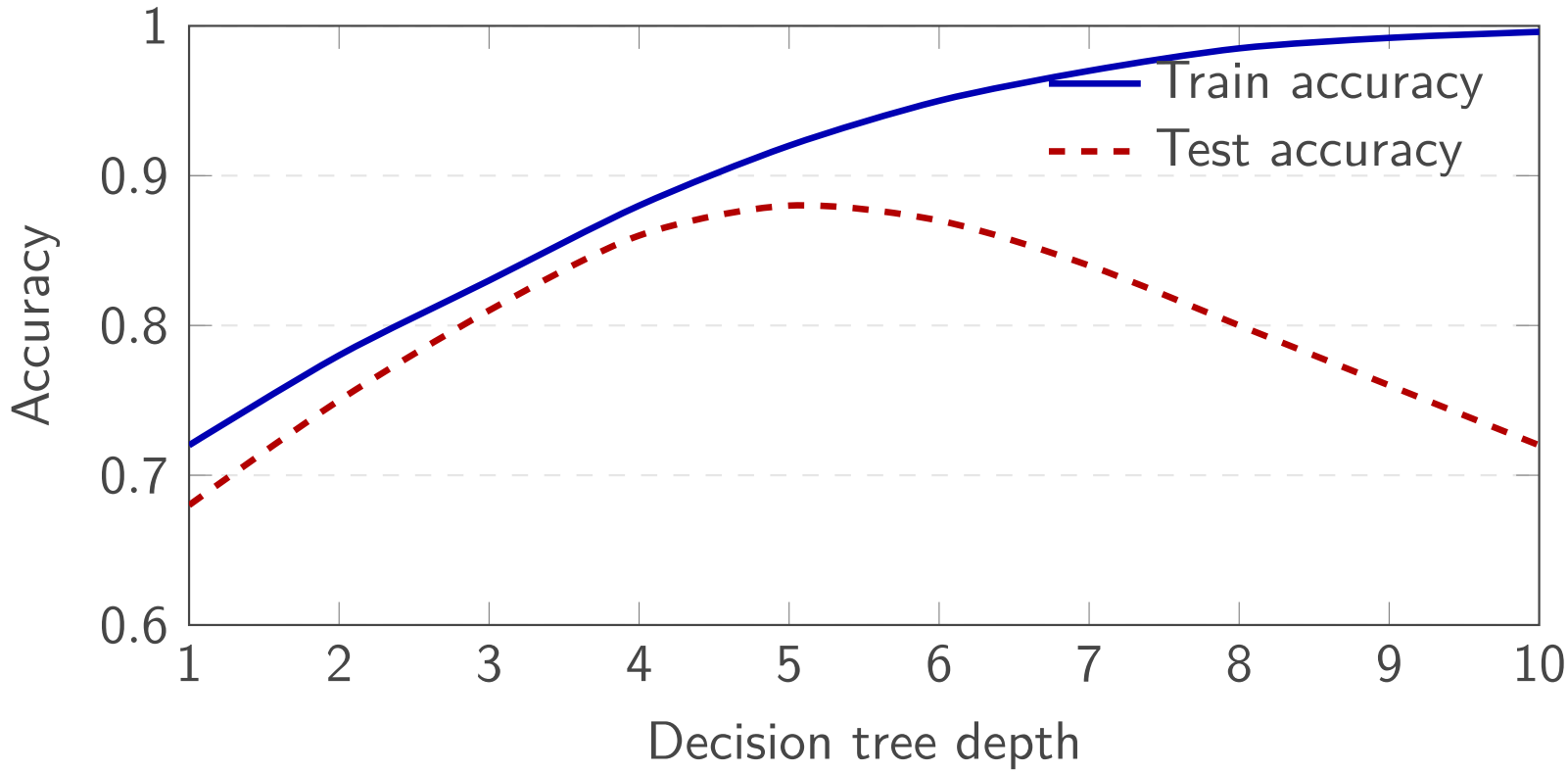


A Good Fit

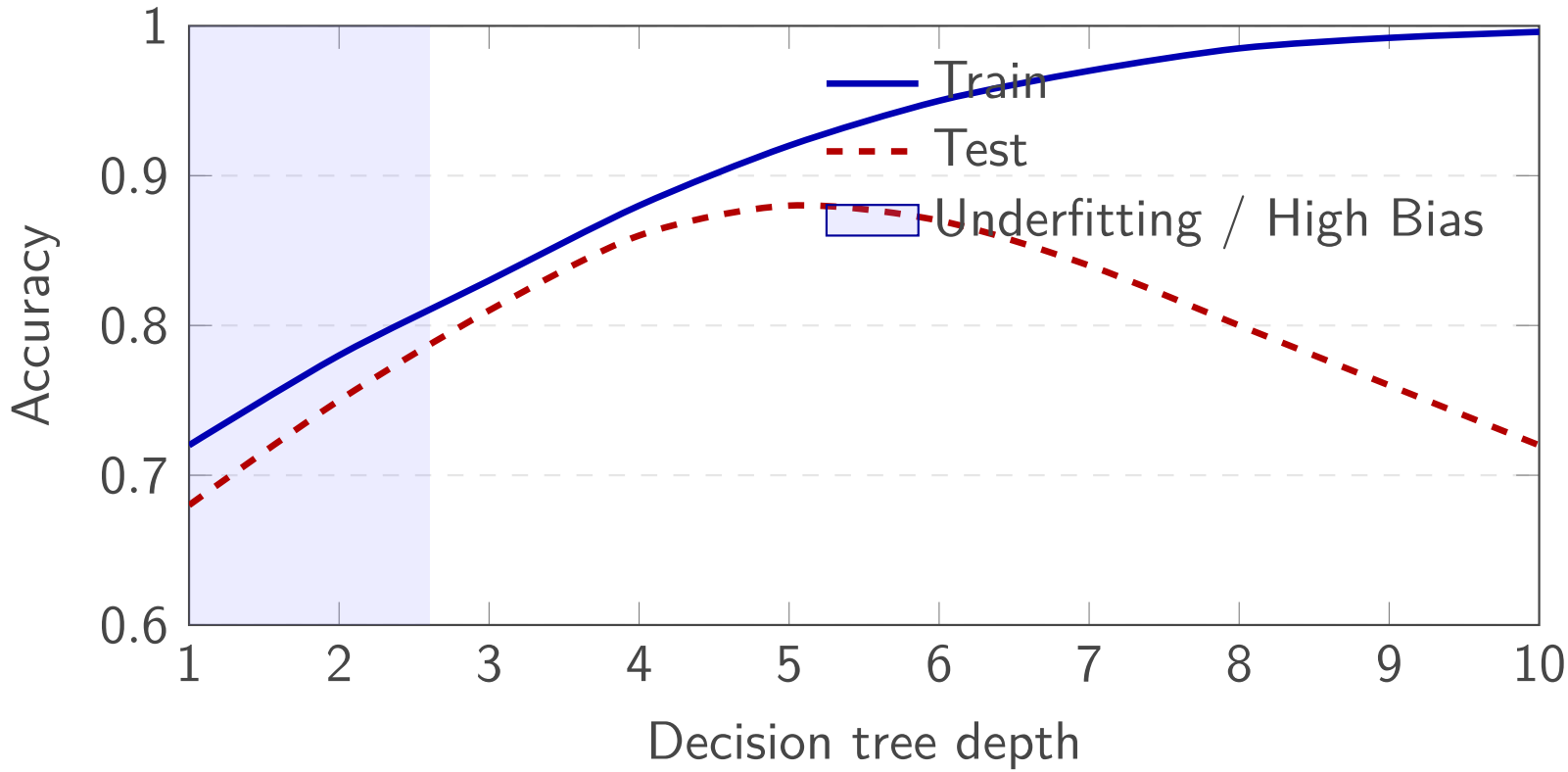

6
5
4
3
2
1
X_2
1
2
3
4
5
6
X_1


X[1] <= 1.5
entropy = 0.964
samples = 36
value = [22, 14]
True
False
entropy = 0.0
samples = 6
value = [6, 0]
X[1] <= 5.5
entropy = 0.997
samples = 30
value = [16, 14]
X[0] <= 1.5
entropy = 0.98
samples = 24
value = [10, 14]
entropy = 0.0
samples = 6
value = [6, 0]
entropy = 0.0
samples = 4
value = [4, 0]
X[0] <= 5.5
entropy = 0.881
samples = 20
value = [6, 14]
entropy = 0.544
samples = 16
value = [2, 14]
entropy = 0.0
samples = 4
value = [4, 0]

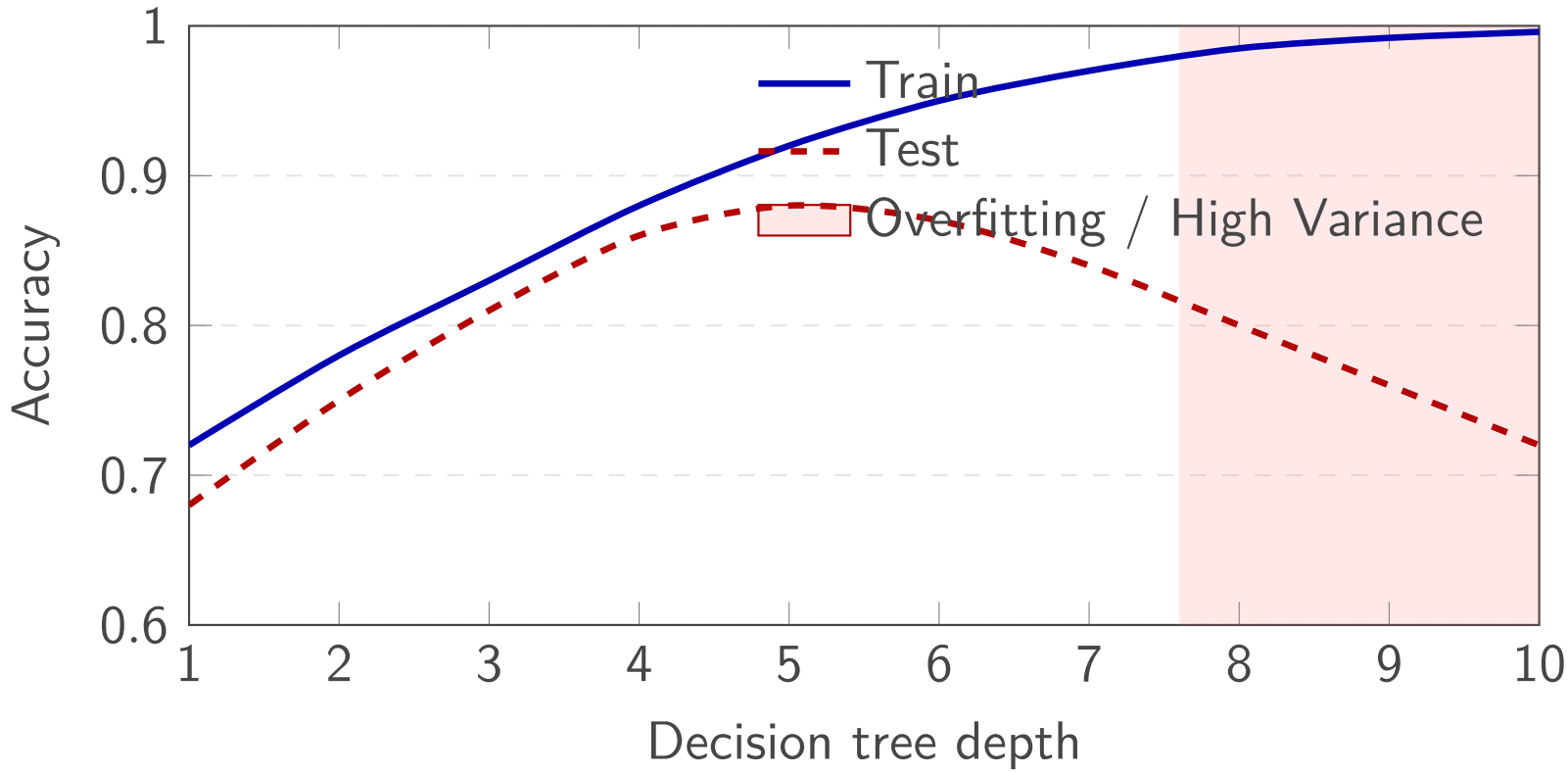
Accuracy vs Decision Tree Depth



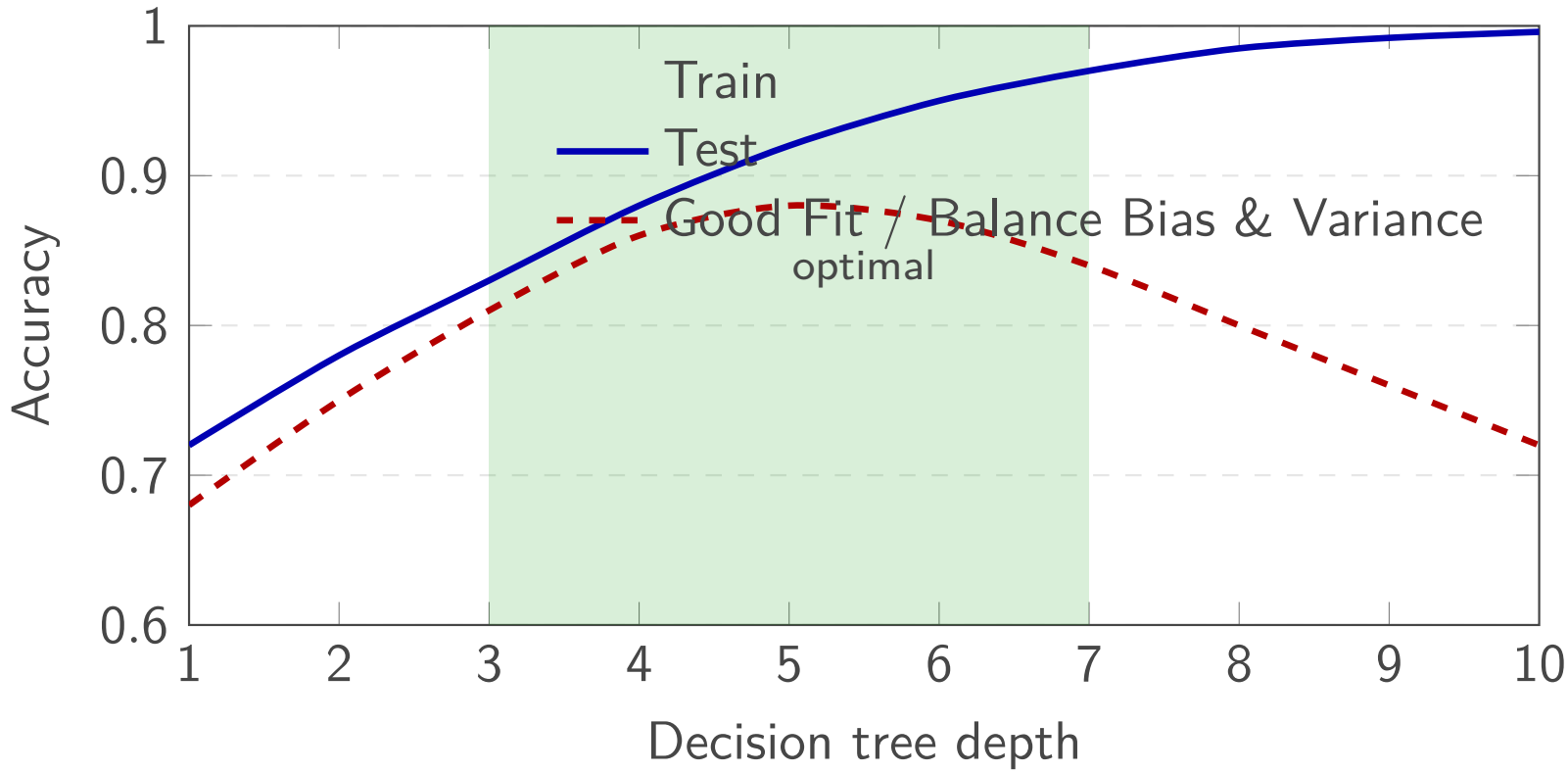
Underfitting (High Bias) Region



Overfitting (High Variance) Region



Good Fit (Sweet Spot) Region



The Fundamental Question: Model Complexity

Important: What We Just Observed

- **Depth 1**: Simple boundary, might miss patterns (underfitting)
- **No depth limit**: Complex boundary, might memorize noise (overfitting)

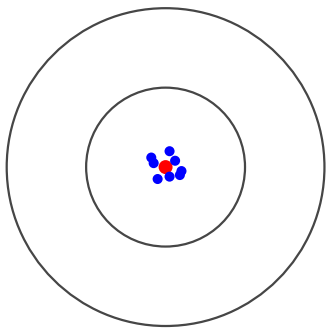
Key Points

This Leads to Three Key Concepts:

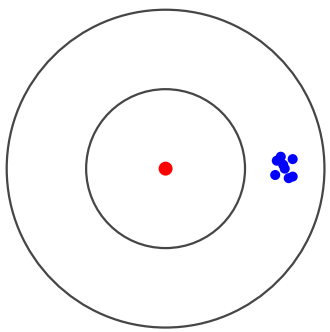
1. **Bias**: How much do our assumptions limit our model's ability to learn?
2. **Variance**: How much does our model change with different training data?
3. **Irreducible Error**: The noise we can never eliminate

The Bias-Variance Tradeoff: We can't minimize both bias and variance simultaneously!

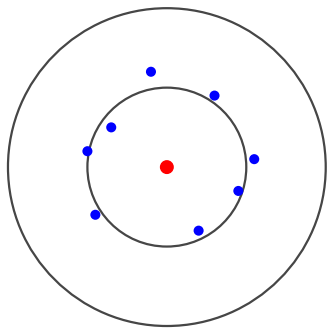
Dartboard Analogy: Four Scenarios



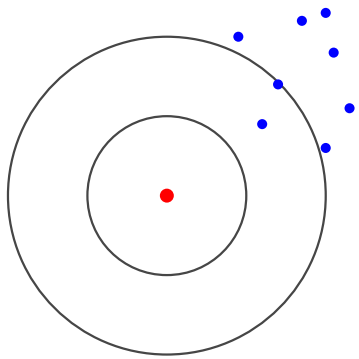
Low Bias, Low Var
Best



High Bias, Low Var
Consistent & wrong



Low Bias, High Var
Inconsistent



High Bias, High Var
Worst

Mathematical Foundation: Bias-Variance Decomposition

Definition: The Fundamental Equation

For any learning algorithm, the expected prediction error can be decomposed as:

$$\text{Expected Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

Where:

- $\text{Bias}^2 = (\mathbb{E}[\hat{f}(x)] - f(x))^2$
  Squared difference between average prediction and true function
- $\text{Variance} = \mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2]$
  Expected squared deviation from average prediction
- $\text{Irreducible Error} = \sigma^2$
  Noise in the data that no model can eliminate

Intuitive Understanding of Each Component

Example: Bias: "Are we systematically wrong?"

- High Bias: Linear model fitting curved data
- Low Bias: Flexible model that can approximate true function
- **Think**: Average error if we could train on infinite datasets

Example: Variance: "Are we consistently wrong?"

- High Variance: Model predictions change drastically with new training data
- Low Variance: Model predictions remain stable across different datasets
- **Think**: How much do predictions fluctuate between training runs?

Key Insight: Both contribute to total error, but reducing one often increases the other!

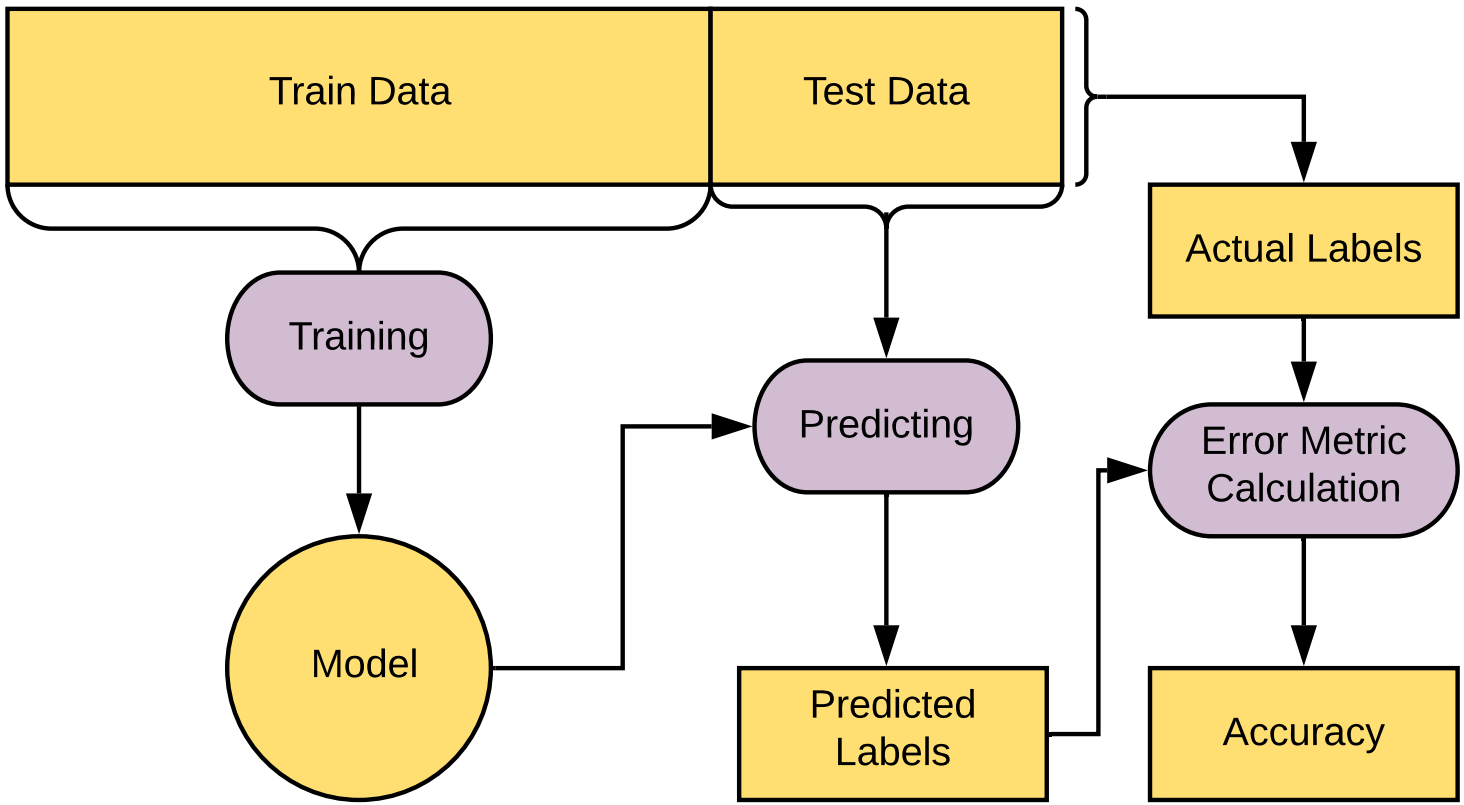
The big question!?

How to find the optimal depth for a decision tree?

The big question!?

How to find the optimal depth for a decision tree?
Use cross-validation!

Our General Training Flow



Example: Training and Evaluation

Step-by-step:

Python Example: Decision Tree with fixed depth

```
# 1. Load dataset
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)
# 2. Train model with fixed hyperparameter
clf = DecisionTreeClassifier(max_depth=3, random_state=42)
clf.fit(X_train, y_train)
# 3. Evaluate
train_acc = accuracy_score(y_train, clf.predict(X_train))
test_acc = accuracy_score(y_test, clf.predict(X_test))
print(f"Max Depth: 3 | Train Acc: {train_acc:.2f} | Test Acc: {test_acc:.2f}")
```

Sample Output:

- Max Depth: 3
- Train Accuracy: 0.98
- Test Accuracy: 0.96

Single Train-Test Split: Setup

Scenario: We trained with `max_depth=3` and got good test accuracy.

Single Train-Test Split: Setup

Scenario: We trained with `max_depth=3` and got good test accuracy.

Our Current Setup

- One fixed train/test split
- Train on training set, evaluate on test set
- Hyperparameter: `max_depth=3` (chosen arbitrarily)

Single Train-Test Split: Setup

Scenario: We trained with `max_depth=3` and got good test accuracy.

Our Current Setup

- One fixed train/test split
- Train on training set, evaluate on test set
- Hyperparameter: `max_depth=3` (chosen arbitrarily)

Example Results

Max Depth	Train Acc	Test Acc
2	0.95	0.94
3	0.98	0.96
4	1.00	0.92

Limitations of a Single Train-Test Split

Why This Can Be Misleading

- Test accuracy depends on how the split was made
- A lucky/unlucky split can overestimate or underestimate performance
- We might miss the **true best hyperparameter**

Limitations of a Single Train-Test Split

Why This Can Be Misleading

- Test accuracy depends on how the split was made
- A lucky/unlucky split can overestimate or underestimate performance
- We might miss the **true best hyperparameter**

Key Insight

Even though `max_depth=3` looks best here, it might not be the best for another random split.

Limitations of a Single Train-Test Split

Why This Can Be Misleading

- Test accuracy depends on how the split was made
- A lucky/unlucky split can overestimate or underestimate performance
- We might miss the **true best hyperparameter**

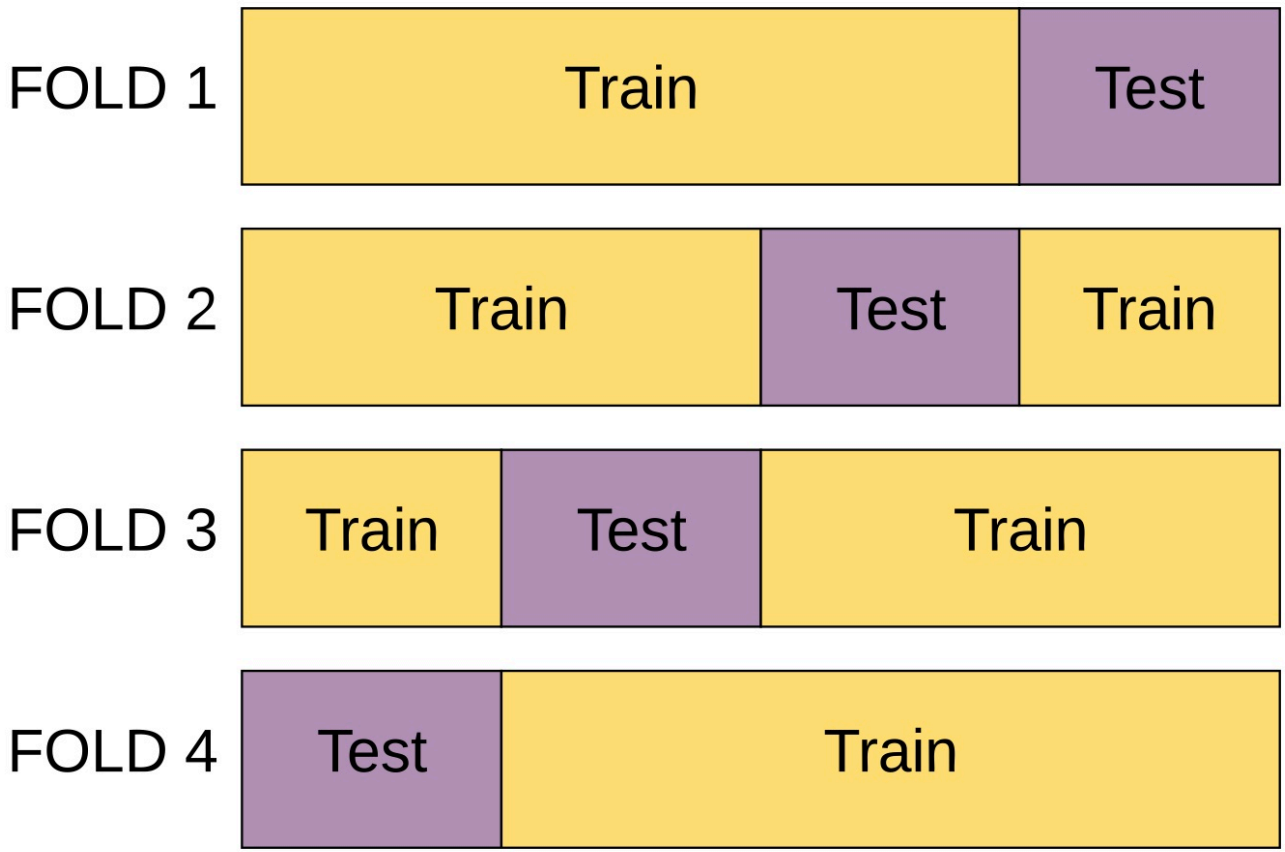
Key Insight

Even though `max_depth=3` looks best here, it might not be the best for another random split.

What We Need

- Evaluate using **all** data for testing
- Avoid overfitting to a single test set

K-Fold cross-validation: Utilise full dataset for testing



Example: K-Fold Cross-Validation (Fixed Depth)

Step-by-step:

Python Example: Decision Tree with fixed depth, using 5-Fold CV

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
acc_scores = []

for train_idx, test_idx in kf.split(X):
    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    clf = DecisionTreeClassifier(max_depth=3, random_state=42)
    clf.fit(X_train, y_train)

    acc_scores.append(accuracy_score(y_test, clf.predict(X_test)))

print(f"Max Depth: 3 | Mean Test Acc: {np.mean(acc_scores):.2f}")
```

Sample Output:

- `Max Depth: 3 | Mean Test Acc: 0.95`

K-Fold Cross-Validation: Insights & Next Steps

What improves here?

- We now test performance on **multiple test sets** (one per fold).
- This gives a **more reliable estimate** of generalization accuracy.

K-Fold Cross-Validation: Insights & Next Steps

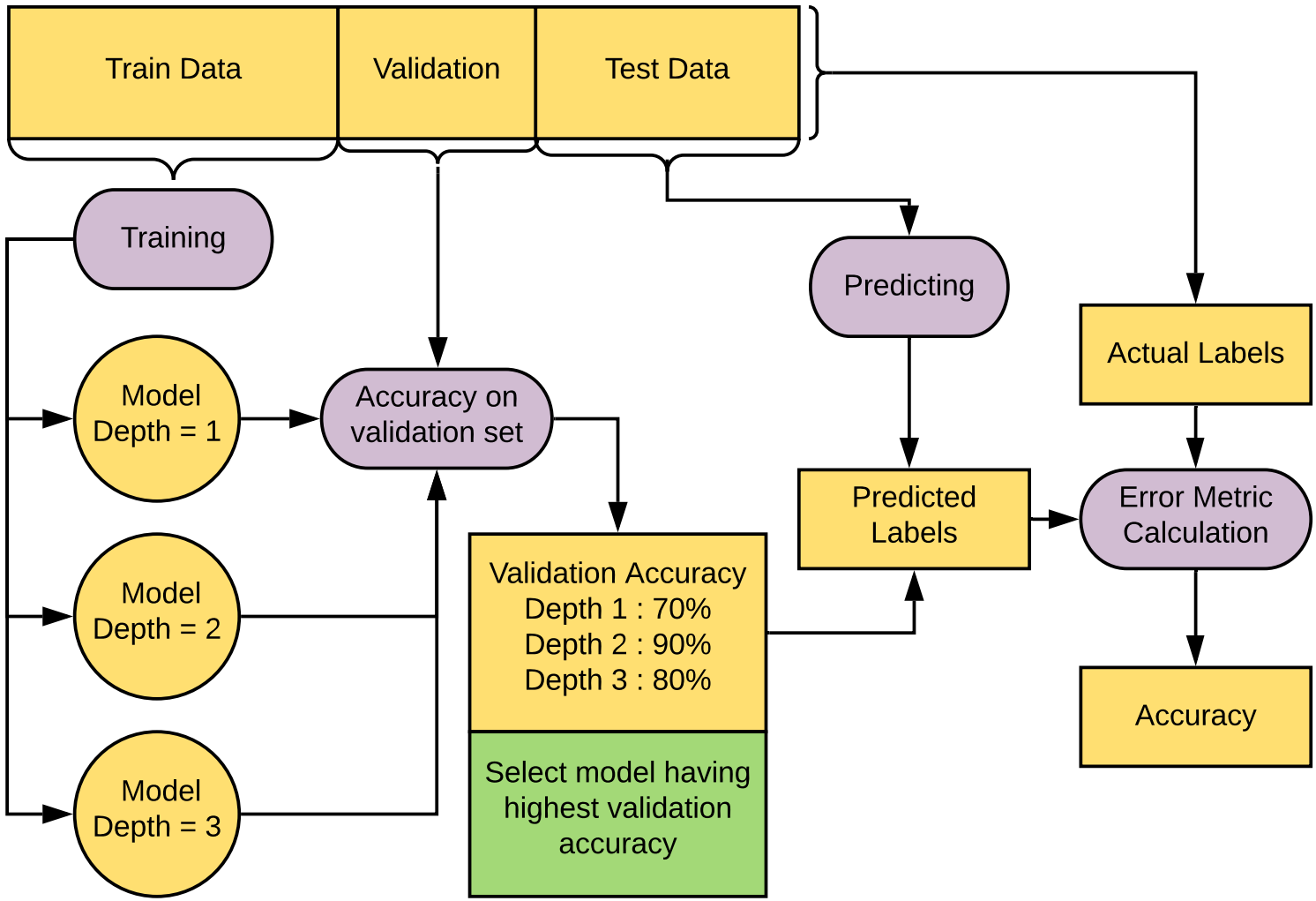
What improves here?

- We now test performance on **multiple test sets** (one per fold).
- This gives a **more reliable estimate** of generalization accuracy.

But... what is the *final model*?

- In K-Fold CV, the model is re-trained for each fold — no single final model exists yet.
- After estimating performance, we can:
  - Re-train on the **entire dataset** with the chosen hyperparameters.
  - Deploy this re-trained model.

The Validation Set



Train/Validation/Test: Hyperparameter Tuning

Idea: Split training set into smaller **train** and **validation** parts.

Python Example: Loop over depths

```
# Split: Train+Val and Test
X_temp, X_test, y_temp, y_test = train_test_split(X, y,
                                                  test_size=0.3,
                                                  random_state=42)
# Split: Train and Val
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp,
                                                  test_size=0.2,
                                                  random_state=42)

best_depth, best_val_acc = None, 0
for depth in [2, 3, 4, 5, 6]:
    clf = DecisionTreeClassifier(max_depth=depth, random_state=42)
    clf.fit(X_train, y_train)
    val_acc = accuracy_score(y_val, clf.predict(X_val))
    if val_acc > best_val_acc:
        best_val_acc, best_depth = val_acc, depth

# Retrain on Train+Val with best depth
final_clf = DecisionTreeClassifier(max_depth=best_depth, random_state=42)
final_clf.fit(X_temp, y_temp)

test_acc = accuracy_score(y_test, final_clf.predict(X_test))
print(f"Best depth: {best_depth}, Test Acc: {test_acc:.2f}")
```

Why Use a Validation Set?

Key Advantages:

- Allows testing **multiple hyperparameters** (e.g., decision tree depth)
- Prevents **overfitting to the test set**
- Gives a clear **criterion for choosing the best model**
- Final test accuracy is now a **realistic estimate of generalization**

Why Use a Validation Set?

Key Advantages:

- Allows testing **multiple hyperparameters** (e.g., decision tree depth)
- Prevents **overfitting to the test set**
- Gives a clear **criterion for choosing the best model**
- Final test accuracy is now a **realistic estimate of generalization**

Limitations:

- Validation set is **only one split** of the data
- May not represent all possible variations in the data

Why Use a Validation Set?

Key Advantages:

- Allows testing **multiple hyperparameters** (e.g., decision tree depth)
- Prevents **overfitting to the test set**
- Gives a clear **criterion for choosing the best model**
- Final test accuracy is now a **realistic estimate of generalization**

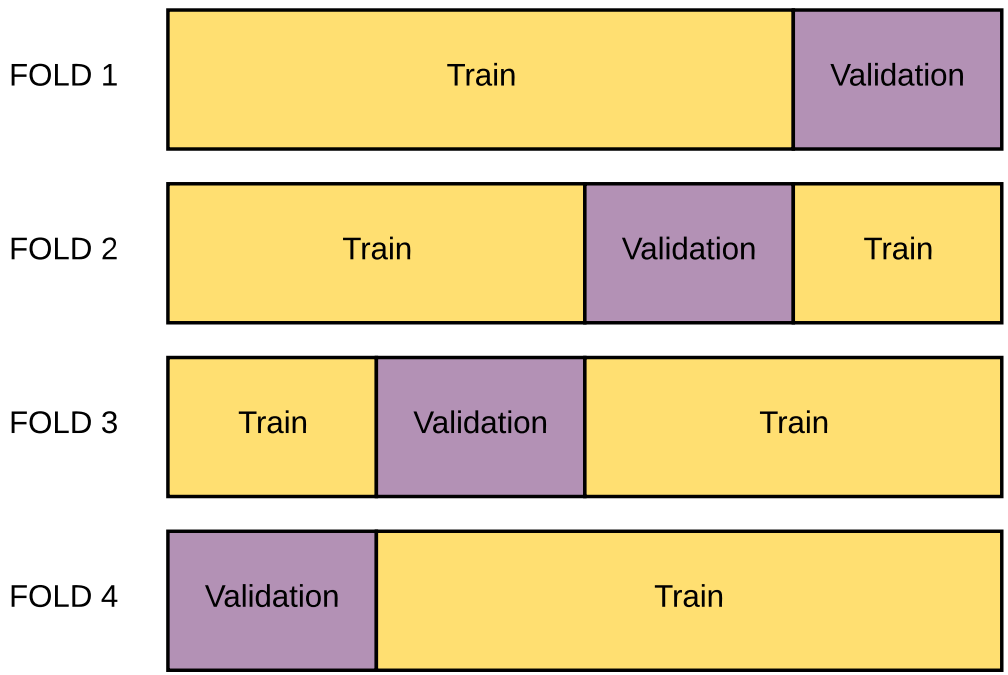
Limitations:

- Validation set is **only one split** of the data
- May not represent all possible variations in the data

Next: How to make use of **all possible validation splits** $\rightarrow$ **Nested Cross-Validation**.

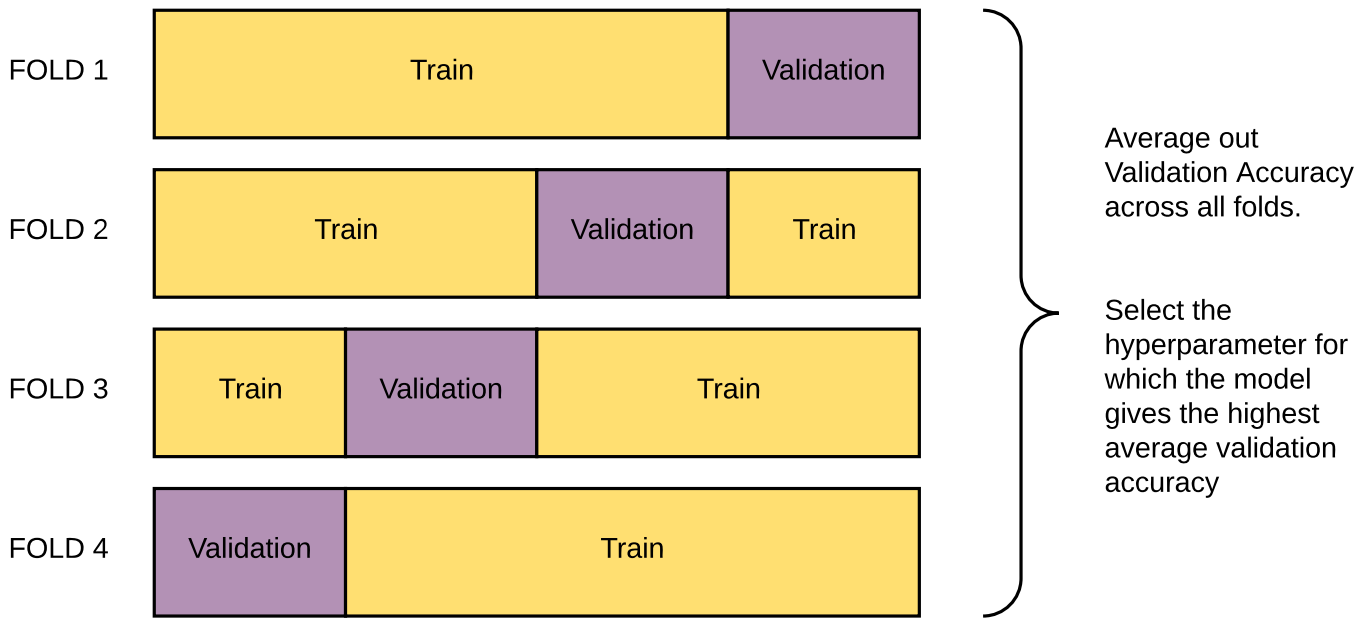
Nested Cross Validation

Divide your training set into k equal parts.
Cyclically use 1 part as "validation set" and the rest for training.
Here $k = 4$



Nested Cross Validation

Average out the validation accuracy across all the folds
Use the model with highest validation accuracy



Nested CV (Inner tuning, Outer testing)

```
outer_kf = KFold(n_splits=5, shuffle=True, random_state=42)
depths = [2, 3, 4, 5, 6]
outer_scores, chosen_depths = [], []

for ot_tr, ot_te in outer_kf.split(X):
    X_tr, X_te = X[ot_tr], X[ot_te]
    y_tr, y_te = y[ot_tr], y[ot_te]
    best_d, best_cv = None, -1

    # Inner CV to pick best depth
    for d in depths:
        scores = []
        for in_tr, in_val in KFold(3, shuffle=True, random_state=42).split(X_tr):
            clf = DecisionTreeClassifier(max_depth=d, random_state=42)
            clf.fit(X_tr[in_tr], y_tr[in_tr])
            scores.append(accuracy_score(y_tr[in_val],
                                         clf.predict(X_tr[in_val])))
        if np.mean(scores) > best_cv:
            best_cv, best_d = np.mean(scores), d

    chosen_depths.append(best_d)
    clf = DecisionTreeClassifier(max_depth=best_d, random_state=42)
    clf.fit(X_tr, y_tr)
    outer_scores.append(accuracy_score(y_te, clf.predict(X_te)))

print(np.mean(outer_scores), chosen_depths)
```

Nested CV: What You Get

Per outer fold:

- A *chosen hyperparameter* (from inner CV)
- An *unbiased test score* (on the outer test split)

Overall:

- Report mean $\pm$ std of outer test accuracy
- *Final model*: retrain on all data with a chosen hyperparameter (e.g., the most frequent/best-average depth from inner CV)

Nested CV: Inner-CV Matrix & Outer Scores

	Inner CV mean accuracy (by depth)						
Outer fold	`d=2`	`d=3`	`d=4`	`d=5`	`d=6`	**Chosen d**	**Outer test acc**
1	0.940	0.960	0.950	0.930	0.900	3	0.95
2	0.950	0.965	0.940	0.920	0.900	3	0.96
3	0.910	0.920	0.935	0.930	0.910	4	0.93
4	0.940	0.955	0.940	0.930	0.900	3	0.95
5	0.945	0.940	0.935	0.920	0.900	2	0.94
Mean across folds	0.937	0.948	0.940	0.926	0.902		$\mathbf{0.95 \pm 0.01}$

Interpretation: Best inner-CV per fold is highlighted; overall best hyperparameter by inner-CV mean is `max_depth` $= 3$. For reporting performance, use the *outer* scores: $\mathbf{0.95 \pm 0.01}$.

Next time: Ensemble Learning

- How to combine various models?
- Why to combine multiple models?
- How can we reduce bias?
- How can we reduce variance?